Exception Handling

Michael B. Spring
Department of Information Science and Telecommunications
University of Pittsburgh
spring@imap.pitt.edu
http://www.sis.pitt.edu/~spring
Overview of Part 1 of the Course

- Demystifying Java: Simple Code
- Introduction to Java
- An Example of OOP in practice
- Object Oriented Programming Concepts
- OOP Concepts -- Advanced
- Hints and for Java
- I/O (Streams) in Java
- Graphical User Interface Coding in Java
- Exceptions and Exception handling

This slide set
Objectives for this unit

• Explain the purpose of exceptions
• Define what an exception is
• Demonstrate how exceptions are triggered
• Demonstrate how exceptions are handled
• Illustrate how exceptions propagate
• Explain when and when not to use exception handling
Concepts

• Exception objects
• Throwing exceptions
• Try / catch / finally blocks
• Propagation
Keywords

- throw
- throws
- try
- catch
- finally
Traditional error handling

• In C:

  if(myFunction() == -1)
  /* handle error */
Drawbacks to traditional error handling

- Inconsistent use of error codes
- Misinterpretation of valid data
- No enforcement of error checking
- Poor readability of code
Exception triggers

- Calling a method that throws an exception
- Use of the keyword throw
- Programmer error (e.g. out-of-bounds array access)
- An internal Java error that is out of your control
public void myMethod() throws MyException
{
    n = getData();
    if(n == null)
        throw new myException();
}
Try / catch blocks

```java
try
{
    // code which may potentially throw an exception
}
catch(MyException e)
{
    // code which handles exceptions of type
    // MyException or any of its subclasses
}
```
try
{
    FileReader input = new FileReader("input.txt");
    input.read();
    input.close();
}
catch (IOException e)
{
    System.out.println("IO error: " + e);
}
try
{
    // code which may potentially throw an exception
}
catch(MyException e)
{
    // perform some type of cleanup

    throw e;
}
Multiple catch clauses

```java
try {
    // access file stream
} catch (FileNotFoundException e1) {
    // handle file not found error
}
catch (IOException e2) {
    // handle all other I/O errors
}
```
Finally blocks

try
{
    // code which may potentially throw an exception
}
catch(MyException e)
{
    // code which handles exceptions of type MyException or
    // any of its subclasses
}
finally
{
    // cleanup code
}
## When to use exception handling

<table>
<thead>
<tr>
<th>Developer of class</th>
<th>User of class</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Body</strong></td>
<td></td>
</tr>
<tr>
<td>Throw the exception (keyword: <code>throw</code>)</td>
<td>Implement a try / catch block (keywords: <code>try</code>, <code>catch</code>)</td>
</tr>
<tr>
<td><strong>Header</strong></td>
<td></td>
</tr>
<tr>
<td>Declare the exception (keyword: <code>throws</code>)</td>
<td>Propagate the exception (keyword: <code>throws</code>)</td>
</tr>
</tbody>
</table>
Important points to remember

- Exceptions derived from RuntimeException are considered unchecked exceptions. All others are referred to as checked exceptions. Only checked exceptions require exception handling.
- A catch clause should try to either handle an error and recover, or clean up and rethrow the exception.
- Order multiple catch clauses in order of the most specific case to the most general.
- Do not resort to exception handling when a trivial test will suffice.
Exercise

• Given: A system of banking accounts developed in exercise # 2 from Thursday’s presentation.
• To Do: Provide additional code that will accept data from the console and output data to a file, catching exceptions as necessary.
• Bonus: Input data from either the data file or console file.